Introducing Development with HBase

Introducing and development with HBase.
In this video, we will go over HBase and
how it defers from an RDBMS.
Then we'll cover Apache Phoenix, HBase and
HDInsight and SQL over HBase.
Apache HBase is a distributed and non-relational,
NoSQL database that runs on top of HTFS.
It's an open source NoSQL database that provides real time
read, write access to those large data sets.
HBase scales linearly to handle large data sets with billions
of rows and millions of columns.
And it easily combines data sources that use a wide variety
of different structures and schemas.
HBase is natively integrated with Hadoop and
works seamlessly alongside other access engine through YARN.
Because it offers strongly consistent read and writes,
HBase is not an eventually consistent data store.
Thus make it very suitable for
tasks such high speed calendar aggregation, and
supports real-time random read and writes.
It provides fault tolerance with applications across
the data center.
Atomic and strongly consistent row level applications.
High availability through automic failover, and
automatic sharding and load balancing of tables.
In HBase, data is stored in tables which have rows and
columns.
This is a terminology overlap with relational databases or
RDBMSs, but this is not a helpful analogy.
Instead, it can be helpful to think of an HBase table as
a multi-dimensional map.
A row in HBase consists of a row key and one or
more columns with values associated with them.
Rows are sorted alphabetically by the row key as they
are stored.
A column in HBase consists of a column family and
a column qualifier, which are delimited by a colon character.
Column families physically co-locate a set of columns and
their values, often for performance reasons.
Each column family has a set of storage properties,
such as whether its value should be cached in memory, how its
data is compressed or its row keys are encoded, and others.
A column qualifier is added to a column family to provide
the index for a given piece of data.
A cell is a combination of row, column family and
column qualifier, and contains a value and
a time stamp which represents the values version.
A time stamp is written alongside each value, and

is the identifier for a given version of a value.
By default, the time stamp represents the time on
the region server when the data was written.
But you can specify a different time stamp value when you put
data into the cell.
The HBase architecture consists mainly of four components,
HMaster, HRegionServer, HRegion and Zookeeper.
First, the client initiates an action that modifies data.
This is currently a call to put, delete, and
increment column value, abbreviated as IMCR at times.
Each of this modifications is wrapped into a key value object
instance and sent over the wire using RPC calls.
The calls are ideally batched to the HRegion server that
serve the affected regions.
Once the payload arrives, the key value is routed to
the HRegion that is responsible for the affected row.
The read and write operations from client into HFile is shown
in this diagram.
Step one, the client wants to write data and in turn first
communicates with the regions server and then the regions.
Step two, regions contact the MemStore for
storing with the associated column family.
Step three, first data stores into the MemStore, where
the data is sorted and after that, it flushes into an HFile.
The main reason for using MemStore is to store data
in the distributed file system based on a row key.
MemStore will be placed in the region server main memory while
HFiles are written into HDFS.
Step four, the client wants to read data from regions.
Step five, in turn, the client can have direct access to
MemStore and it can request the data.
In step six,
the client approaches HFiles to get the data.
The data is fetched and retrieved by the client.
MemStore holds in memory modifications to the store.
To reiterate,
the HBase architecture consists of mainly four components.
HMaster, HRegion Server, HRegions, and Zookeeper.
HMaster is the implementation of the master server in the HBase
architecture.
It acts like a monitoring agent to monitor all region server
instances present in the cluster and acts as an interface for
all the metadata changes.
HRegionsServers, when a region server receives read and
write requests from the client, it assigns the request to
a specific region, where the actual column family resides.
However, the client can directly contact the HRegionsServers.
And there is no reason for
HMaster mandatory permissions to the client regarding
communication with HRegionsServers.

The client requires HMaster's help when operations related to metadata and schema changes are required.
HRegions are the basic building elements of the HBase cluster that consist of the distribution of tables and are comprised of column families.
It contains multiple stores, one for each column family.
It also consists of mainly two components, which are Memstore and HFile.
The class which implements the write-ahead Log or WAL is called HLog.
There is only one instance of the HLog class, which is one per HRegion server.
When an HRegion is instantiated, the single HLog is passed on as a parameter to the constructor of HRegion.
The data lives in StoreFiles and there are multiple StoreFiles per store.
StoreFiles are implemented as HFiles and permit random read write access.
Memstore is an in-memory cache, one per store.
All changes are first stored in the cache, and then persistently written to the store file asynchronously.
This slide shows the hierarchy of objects in HBase Regions from top to bottom.
First, there's the HBase Table present in the HBase cluster.
Next, we have the HRegions for the presented tables.
Following that is the store, which stores per ColumnFamily for each region of the table.
And after that is the MemStore for each store within each region of the table.
It stores data before flushing into HFiles.
Write and read performance will increase because of sorting.
Also at this level, are the StoreFiles for each region of the table.
Finally, there are blocks present inside the StoreFiles.
Apache HBase distributes load through regions and manages regions splitting.
It stores rows of data and tables and tables are split into chunk of rows called regions.
Those regions are distributed across the cluster, hosted, and made available to client processes by the region's server process.
A region is a continuous range within the key space, meaning all rows in the table that sort between the region's start key and end key are stored in the same region.
Regions are non-overlapping, meaning a single row key belongs to exactly one region at any point in time.
The benefits of splitting data into regions are fast recovery, when a region fails.
Load balancing when a server is overloaded as regions may be moved between servers.

Splitting is done asynchronously while data is being read.
And splitting is automated,
requiring no manual intervention.
HBase is written in Java and has a Java native API.
Therefore, it provides programmatic access to the data
manipulation language or DML.
The Get command and the get method of HTable class are used
to read data from a table on HBase.
Using the Get command,
you can retrieve a single row of data at a time.
Use the scan command to fetch the entire row or
only rows matching specified rowkeys.
You can insert and add new data or
update existing cells using the put command or add command.
The delete class is used to perform delete operations on
a single row.
To delete an entire row,
instantiate a delete object with the row to delete.
This is a sample customer data file in HBase
with tab separated fields.
And this is a sample orders data file in
HBase also with tab separated fields.
Applications can run on top of HBase by using
it as a datastore.
Examples include Phoenix, OpenTSDB, Kiji, and Titan.
Applications can also integrate with HBase.
Examples include Hive, Pig, Solr, Storm, Flume, Impala,
Ganglia, and Drill.
HBase adds transactional capabilities to
the Hadoop ecosystem.
Allowing customers to perform fast record updates and
lookups on large datasets in Azure Blobs.
As a distributed database, HBase was architected to scale as load
and performance demand increases.
Thus, HBase is ideal for customers who want to do
transactions of millions to billions of rows of data.
At general availability,
HBase will support up to 500 terabytes in Azure Blobs.
However, HBase wasn't built to replace all scenarios of
a standard RDBMS, as it lacks features like an optimizer,
secondary indexes, advanced query languages, etc.
There are some difference between RDBMSs and HBase.
An RDBMS database is row-oriented and suitable for
online transaction processes or OLTP.
Such databases are designed for a small number of rows and
columns.
HBase is a column-oriented database and suitable for
online analytical processing or OLAP.
Column-oriented databases are designed for huge tables.
RDBMSs use SQL queries for reading records from tables.

There is no built in support for HBase.
This uses APIs and MapReduce for
accessing data from HBase tables.
HBase has to use APIs for accessing the data.
RDBMS has ACID support,
there is no transaction guarantee in HBase.
ACID over single row or a adjacent row groups only.
Joins operation are possible in RDBMS, but not in HBase.
You can use single row indexes only in HBase.
Apache Phoenix is an open source, massively parallel
relational database engine supporting OLTP for
Hadoop using Apache HBase as its backing store.
It's an SQL skin over HBase delivered as an client-embedded
JDBC driver, targeting low-latency queries.
HBase is designed to handle billions of rows and
millions of columns.
However, using it as a relational database,
where you would store your data normalized,
split into multiple tables, is not easy.
Phoenix provides a JDBC driver that hides the intricacies of
the NoSQL store.
Enabling customers to create, delete and alter SQL tables,
views, indexes and sequences, upsert and
delete rows singly and in bulk, and query data through SQL.
Phoenix compiles queries and
other statements into native NoSQL store APIs.
Rather than using MapReduce, enabling the building of low
latency application on top of NoSQL stores.
Apache Phoenix is fully integrated with other Hadoop
products, such as Spark, Hive, Pig, Flume, and MapReduce.
Apache Phoenix takes your SQL query, compiles it into a series
of HBase scans, and orchestrates the running of those scans to
produce regular JDBC result sets.
Direct use of the HBase API along with coprocessors and
custom filters results in performance on the order of
milliseconds for small queries or seconds for
tens of millions of rows.
All standard SQL query constructs are supported,
including SELECT, FROM, WHERE, GROUP BY, HAVING, ORDER BY, etc.
It also supports a full set of DML commands,
as well as table creation and
version incremental alterations through our DDL commands.
Here's a list of what is currently not supported,
relational operators such as intersected minus, and
miscellaneous built-in functions.
There are three main components of Apache Phoenix.
Phoenix Includes a JDBC driver as a client side driver
that allows users to query and manage SQL tables directly.
Then there's the query execution engine that translates queries
in other statements to native NoSQL API calls,

instead of using MapReduce.
Thus, enabling faster applications on top of NoSQL stores.
It pushes down operations onto the cluster for maximum parallelism.
Also there is the metadata repository that provides type access to data stored in HBase tables.
HDInsight HBase is offered as a managed cluster that is integrated into the Azure environment.
The clusters are configured to store data directly in Azure Blob Storage, which provides low latency and increased elasticity in performance and cost choices.
This enables customers to build interactive websites that work with large data sets, to build services that store sensor and telemerty data for millions of endpoints, and to analyze this data with Hadoop jobs.
The HDInsight implementation leverages the scale-out architecture of HBase to provide automatic sharding of tables, strong consistency for reads and writes, and automatic failover.
Performance is enhanced by in-memory caching for reads and high throughput streaming for writes.
Virtual network provisioning is also available for HDInsight HBase.
Data can be managed in HBase by using the create, get, put, and scan commands from the HBase shell.
Data is written to the database by using put and read by using get.
The scan command is used to obtain data from multiple rows in a table.
Data can also be managed using the HBase C# API, which provides a client library on top of the HBase REST API.
An HBase database can also be queried by using Hive.
Coprocessors are also available, which allow data processing in the nodes that host the database.
The canonical use case for which big table and by extension HBase was created was web search.
Search engines built indexes that map terms to the web pages that contain them.
But there are many other used cases that HBase is suitable for, several of which are itemized in this section.
First, there is the key value store.
HBase can be used as a key value store and it is suitable for managing message systems.
Facebook uses HBase for their messaging system and it is ideal for storing and managing Internet communications.
Web table uses HBase to search for and manage tables that are extracted from web pages.
Next is sensor data, HBase is useful for capturing data that is collected incrementally from various sources.
This includes social analytics, time series,

keeping interactive dashboards up to date with trends and counters, and managing audit log systems.
Examples include Bloomberg Trainer Terminal and the Open Time Series Database, OpenTSDB, which stores and provides access to metrics collected about the health of server systems.
Then there's real-time query.
Phoenix is a SQL query engine for Apache HBase.
It is accessed as a JDBC driver and it enables querying and managing HBase tables by using SQL.
Finally, there is HBase as a platform.
Applications can run on top of HBase by using it as a data store.
Examples include Phoenix, OpenTSDB, Kiji and Titan.
Applications can also integrate with HBase.
Examples of these include Hive, Pig, Solr, Storm, Bloom, Impala, Spark, Ganglia, and Drill.